Wrocław, 16 kwietnia 2024

Paradygmaty Programowania Obiektowego

Laboratorium 2

Dziedziczenie i enkapsulacja

prowadzący: *mgr inż. Marta Lampasiak*

1 Wprowadzenie

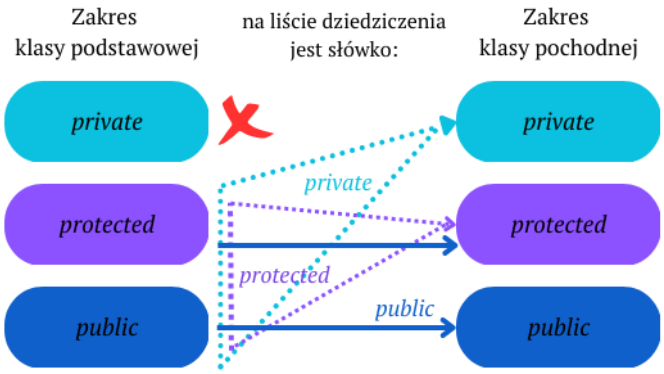
Celem laboratorium jest poznanie i przećwiczenie użycia mechanizmu dziedziczenia.

Dziedziczenie jest techniką umożliwiającą tworzenie nowych klas na bazie już istniejących. Klasa dziedzicząca (klasa pochodna) po innej klasie (klasie podstawowej, bazowej) przejmuje jej składowe oraz dodaje swoje.

Istnieją trzy typy dziedziczenia, które różnią się dostępem do odziedziczonych składowych w klasie pochodnej. Warto zaznaczyć, że składnik prywatny klasy podstawowej jest dziedziczony przez klasę pochodną, ale nie jest dla niej bezpośrednio dostępny. Natomiast istnieje mechanizm, który określa w jaki sposób klasa pochodna ma odziedziczyć składnik **nieprywatny**, czyli *public* lub *protected*. Odbywa się to poprzez umieszczenie na liście pochodzenia specyfikatora dostępu.

```
class Pochodna : public Bazowa // lista dostepowa
{
  // specyfikator dostepu na liscie dostepowej : public
}
```

Skutki użycia specyfikatorów dostępu w ramach dziedziczenia pokazuje rysunek 1.



Rysunek 1: Dziedziczenie – specyfikatory dostępu

Przedstawiony graf należy odczytywać w następujący sposób: Jeżeli na strzałce grafu, czyli w implementacji na liście dostępowej znajduje się specyfikator dostępu:

- `private` — wszystkie składowe publiczne i chronione klasy bazowej stają się prywatne w klasie pochodnej,
- `protected` — składowe publiczne klasy bazowej stają się chronione w klasie pochodnej, a chronione pozostają chronione,
- `public` — wszystkie składowe publiczne klasy bazowej są publiczne w klasie pochodnej, chronione pozostają chronione.

Brak strzałki przy obszarze *private* w klasie podstawowej oznacza (jak zostało już wcześniej wspomniane), że zmiana rodzaju dostępu nie następuje. Odziedziczone elementy mają nadal dostęp *private*, ale w zakresie ważności klasy podstawowej. Ten zakres jest zasadniczo niedostępny z zakresu klasy pochodnej.

W przypadku typu dziedziczenia *public* nieprywatne składniki klasy podstawowej, czyli składniki *protected* i *public* są dostępne w zakresie klasy pochodnej **bezpośrednio**. Tutaj można zauważyć różnicę pomiędzy *private* a *protected*. Składnik *protected* jest zastrzeżony dla klas pochodnych. W związku z tym „z zewnątrz" składnik *protected* będzie niedostępny i traktowany jako *private*, ale dla klas pochodnych od danej klasy będzie on się zachowywał jak *public*.

Dziedziczeniu w języku C++ nie podlegają: konstruktory, destruktory oraz operator przypisania. W związku z czym należy pamiętać, że w celu inicjalizacji odziedziczonych pól klasy podstawowej tworzy się osobny konstruktor, konstruktor klasy pochodnej, który wywołuje konstruktor klasy bazowej.

```
class Bazowa
{
private:
    int b_;

public:
    Bazowa() : b_{0} {};  // konstruktor domyslny z lista inicjalizacyjna
};

class Pochodna : public Bazowa
{
private:
    double p_;
public:
    // wykorzystanie konstruktora klasy bazowej w celu zainicjalizowania
    // pola odziedziczonego przez klase pochodna
    Pochodna() : Bazowa(), p_{0.0} {}
};
```

Jeżeli nie chcemy, by dana klasa była użyta do tworzenia klasy pochodnej, wystarczy zastrzec to, umieszczając w pierwszej linijce definicji danej klasy słowo *final* (finalny, ostateczny).

```
class Bazowa final
{
    //...
};
```

2 Zadanie

Zadanie polega na utworzeniu klasy bazowej oraz dwóch klas pochodnych. Każda z klas posiada konstruktor, destruktor oraz odpowiednie metody. Należy przeprowadzić testy utworzonych klas oraz odpowiedzieć na zadane pytania w komentarzach w kodzie (odpowiedzi należy uzasadnić). Stanowią one część oceny. Poniżej znajduje się lista poszczególnych kroków, jakie należy wykonać wraz z wyszczególnieniem progów ocen. W celu uzyskania maksymalnej oceny należy wykonać wszystkie kroki.

Polecenia *ocena 3.0*

1. Wykonaj następujące polecenia:

   - Napisz klasę `Animal` zawierającą pola: *prywatne*: weight_ (zmienna typu *float*), color_ (zmienna typu *string*)

   oraz funkcje składowe o etykiecie dostępowej *public*:

   - konstruktor domyślny (domniemany), który poprzez listę inicjalizacyjną nadaje polom wartości: 0.0, "none" oraz wyświetla tekst na standardowym wyjściu: *„Konstruktor domyslny – klasa podstawowa Animal"*,

- Destruktor wypisujący napis: *"Destruktor – klasa podstawowa Animal*,
- metodę `eat()` wyświetlającą tekst "I can eat! – klasa podstawowa Animal".
- tzw. metody *Getters* – umożliwiające dostęp do pól prywatnych klasy, pobieranie ich wartości, czyli stwórz funkcje: `get_weight()` oraz `get_color()`. Pamiętaj, że są to tzw. funkcje dostępowe, przez co powinny posiadać odpowiednie słówko kluczowe.

2. Utwórz klasę pochodną `Tiger` klasy podstawowej `Animal` o specyfikatorze dostępu *public* na liście pochodzenia. Stwórz metody o etykiecie dostępowej *public*:
   - konstruktor domyślny wykorzystujący w liście inicjalizacyjnej kosntruktor klasy podstawowej oraz wypisujący tekst: *"Konstruktor domyślny – klasa pochodna Tiger"*,
   - Destruktor wypisujący napis *"Destruktor – klasa pochodna Tiger"*,
   - metodę `roar()` wypisującą napis *"rrrr"*.
3. W pętli głównej programu:
   - Utwórz obiekt klasy `Tiger`.
   - Dla stworzonego obiektu wywołaj metody klasy `Animal`: `eat()`, `get_color()`, `get_weight()`.
   - Dla stworzonego obiektu wywołaj metodę `roar()`.
4. Odpowiedz na PYTANIA:
   (a) Działanie którego konstruktora zaobserowałeś i dlaczego?
   (b) Czy udało się wywołać metody klasy podstawowej dla obiektu klasy pochodnej? Dlaczego?
   (c) Czy zauważyłeś działanie destruktora? Jeżeli tak, to jakiego i dlaczego?

Polecenia *ocena 4.0*

Do klasy podstawowej **dodaj składową** *protected* o nazwie `type_` (zmienna typu *string*). Następnie do klasy *Tiger* **dodaj metody** *public*:

- `set_type` – ustawia wartość pola `type_` na wartość przekazaną w argumencie.
- `display_information` – wyświetlającą napis *"It's a "* oraz wartość pola `type_`.

Korzystając z odpowiednich metod, w pętli głównej programu, nadaj wcześniej utworzonemu obiektowi typ *"mammal"* (ssak) oraz wyświetl o nim informacje. W komentarzu odpowiedz na PYTANIE: Wyjaśnij dlaczego istnieje możliwość nadania wartości polu w klasie podstawowej poprzez metodę klasy pochodnej?

Polecenia *ocena 5.0*

Utwórz klasę `Bear`, która jest klasą pochodną klasy `Animal` z dziedziczeniem *public* i zaimplementuj dla niej:

- pola prywatne: `weight_` oraz `color_`.
- Konstruktor wieloargumentowy z wykorzystaniem listy inicjalizacyjnej, wyświetlający tekst: *"Konstruktor wieloargumentowy - klasa pochodna Bear"*.
- Destruktor wypisujący napis *"Destruktor – klasa pochodna Bear"*,
- metody *Getters* umożliwiające dostęp do pól prywatnych o nazwach: `get_weight()` oraz `get_color()`.

Utwórz w głównej pętli programu obiekt klasy `Bear` i nadaj mu wartości poprzez stworzony konstruktor. Wykonaj poszczególne kroki (na wszelkie pytania odpowiadaj w komentarzach):

- Wyświetl wartości pól prywatnych – wykorzystaj „gettery”. PYTANIE: Jakie wartości zostały wyświetlone?
- Wywołaj „gettery” na rzecz tego samego obiektu w głównej pętli programu, ale tym razem dopisz przed ich wywołaniem deklarację dostępu, czyli np. `Animal::get_weight()`. PYTANIE: Jakie wartości teraz uzyskałeś?
- Na podstawie uzyskanych rezultatów wyjaśnij/odpowiedz na PYTANIE: co się dzieje w momencie, gdy klasa pochodna i klasa podstawowa posiadają pola o tych samych nazwach?